2010

Web-Dinar: Web Based Diagnosis of Network and Application Resources in Disaster Response Systems

Kartik Deshpande
University of Massachusetts Amherst

WEB-DINAR:

WEB BASED DIAGNOSIS OF NETWORK AND APPLICATION RESOURCES IN DISASTER RESPONSE SYSTEMS

A Thesis Presented

by

KARTIK DESHPANDE

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

May 2010

ELECTRICAL AND COMPUTER ENGINEERING

WEB-DINAR:

Web Based Diagnosis of Network and Application Resources in Disaster response systems

A Thesis Presented

by

KARTIK DESHPANDE

Approved as to style and content by:

_____
Aura Ganz, Chair

_____
C. Mani Krishna, Member

_____
Tilman Wolf, Member

_____
C. V. Hollot, Department Head
Electrical & Computer Engineering

ACKNOWLEDGEMENTS

It gives me immense pleasure to present my thesis work of DiNAR guided by Prof. Aura Ganz. It has been a great learning experience working with her in Multimedia Networking Lab. Working on live projects like Diorama have given me a great understanding of the challenges involved. Apart from the technical knowledge I will also take with me the professionalism she taught us when approaching any project.

I am also very thankful to Prof C. Mani Krishna and Prof. Tilman Wolf for agreeing to be on my thesis committee and providing adequate inputs.

Lastly all this would not have been possible without the support of my parents, wife (Rashmi) and brother. It made this journey much easier than it could have been.

`

ABSTRACT

WEB-DINAR:
WEB BASED DIAGNOSIS OF NETWORK AND APPLICATION RESOURCES IN DISASTER RESPONSE SYSTEMS

KARTIK DESHPANDE, B.E., PESIT BANGALORE

M.S.E.C.E, UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Aura Ganz

Disaster management and emergency response mechanisms are coming of age post 9/11. Paper based triaging and evacuation is slowly being replaced with much advanced mechanisms using remote clients (Laptops, Thin clients, PDAs), RFiDs etc. This reflects a modern trend to deploy Information Technology (IT) in disaster management. IT elements provide a great a deal of flexibility and seamlessness in the communication of information. The information flowing is so critical that, loss of data is not at all acceptable. Loss of data would mean loss of critical medical information portraying the disaster scenario. This would amount to a wrong picture being painted of the disaster incident. This basic idea led to the motivation of DiNAR (Diagnosis of Network and Application Resource). The aim of DiNAR was to remotely monitor all the components of the deployed system infrastructure (Remote clients, Servers) and if there is a fault in the infrastructure (Hardware, Software or Communication) DiNAR captures the fault alarm and do an event correlation to find the source of the problem.

The biggest challenge that lies here is the fact that the entities we are trying to monitor are scattered around in the Internet. Traditional network management techniques always assume that the network is within administrative control and every device we monitor is

`

easily reachable on demand. But the ad-hoc scenario of deployment of disaster management systems makes this task non trivial.

DiNAR has been designed with an aim to work with any application which has its infrastructure elements scattered in the Internet space. DIORAMA (A real time disaster management system) represents a new series of applications (especially in medical field) where the deployment of network infrastructure is scattered around with Internet being the backbone connector. Another such example is the Intel® Health Guide PHS6000 [1], which is used in patient monitoring in homes. This thesis work uses DIORAMA as a case study application used to prove the concept of DiNAR.

`

TABLE OF CONTENTS

Page

`

`

LIST OF TABLES

`

LIST OF FIGURES

`

CHAPTER 1

INTRODUCTION

"Disasters are events that disrupt the normal functioning of the economy and society on a large scale" [7]. One word to describe disaster events is *complex*. This complexity comes from the sudden and abrupt nature of disasters. It disrupts the normal socio-economic setup. Thus it is very essential that disaster response management is handled by a specialized set of trained people in handling such events. It also needs a very thoughtful and streamlined process. Emergency Medical Services (EMS) experts believe that there are four major phases in disaster: "mitigation, preparedness, response, and recovery" [7]. Of these four phases, the response and recovery from disaster pose the biggest challenges for EMS officials. Disaster management has been a very vibrant area of research from technological perspective off late. Till recently, it was considered a much localized phenomenon where the efforts were based more on local resources and organizations [7].

Disaster response management has seen significant growth in terms of technology and processes after 9/11. One of the major reasons to this can be attributed to the need for good evacuation and victim tracking mechanism, the lack of which contributed to significant casualties during 9/11 and previous disasters. The traditional EMS disaster evacuation process involved paramedics using paper tags with different color codes to represent the victim condition [2]. The victims were tagged with these paper tags with their corresponding color code. The paramedic then establishes contact with a command center using the available communication mechanisms like satellite phones or other

`

media to report the victims' statistics. This primitive method is now being replaced by use of modern sensing technology and Internet. This advancement in technology adds speed and ease of managing the information generated at the disaster site.

1.1. Example Of Information Technology (IT) In Disaster Management:

Figure 1.IT In Disaster Management [12]

A normal setup of disaster management system consists of a “Disaster Site” from where emergency medical information is being collected and a “Remote Site” which is the information sink for all the data.

Figure 1 shows an emergency response network with its main components such as

1) Wireless network devices deployed at the disaster site (used for both wireless local area network communication and cellular communication). Such devices interconnect

`

the devices in the disaster site as well as relay the information from the disaster site to the remote site.

2) Computing and sensing devices at the disaster site such as Remote clients, RFID readers. Such devices collect the relevant information from the disaster site, process it and transmit it to the remote servers through the

This setup can change depending on the technology used to relay the information to the remote site from the disaster site. If each device is empowered with Interconnectivity using 3G/GPRS or other cellular technologies, the topology will look slightly different. But the method of information collection across the two sites still remains the same.

1.2. Motivation for Dinar:

Modern day disaster management has Information Technology (IT) as a major player in it. The activities which involve IT in this process are, identifying the EMS and related resources, establishing connectivity with these resources and deploying them where needed. This is then followed by coordinating the activities and providing communication between various geographically separated locations [7]. The National Academic Report (2007) [7] on use of IT for emergency response management gives us some of the examples of application of IT in disaster response management. Some of these use modern sensing technologies like RFID, sensor networks coupled with wireless networks, Internet etc to provide an end to end solution for disaster management. Thus forming a mini overlay network with the Internet as its backbone. One of the primary motives of applying IT to disaster management is to provide quick communication methodology and faster response time. It also helps in quickly building

`

a high level view of the disaster scene. On a broad sense, it can be visualized as building a seamless information flow from the site of disaster to the command centers where actual decisions to act are taken.

Although IT makes disaster response management quick and simple, the biggest question mark on its practical applicability lies in its reliability. It uses various elements of IT like:

1. Internet
2. Remote clients and Servers
3. Wireless networking devices
4. Etc.

And all these elements have inherent potential to fail or to be mis-configured. Failure to operate of any entity in any section of the information flow could lead to loss of the critical information. This information lost would reflect in a wrong perception being printed about the disaster. This is a potentially very dangerous situation. Use of technology to solve problem can be potentially disastrous if the reliability is not properly addressed. But we need to understand that the IT elements which constitute disaster management systems are very volatile. Below are few of the scenarios of failure of components which are common:

1. Application crashes
2. Device power outage
3. Wireless network access issues
4. Internet outages

`

Thus any of the above scenarios would lead to loss of information which is getting generated dynamically from the disaster site. It is very essential to understand the criticality of this information which is getting lost. This information is getting generated seamlessly from the disaster site and represents the state of victims. For example, an application reading data from RFID readers could crash and lead to complete blacking out of the RFID reader. This is not a hardware fault or a communication problem. Yet the information is lost. This would mean loss of vital victim statistics. Hence every piece of information being carried is very critical and there is a vulnerability of losing it.

Reliability is a big concern in a volatile system like this, or for that matter any enterprise network. In practical world, failures cannot be avoided. But the robustness of a system depends on how quick the failure is detected and recovered from. Traditional enterprise networks invest heavily on softwares that do automated network management. Despite of the sophisticated softwares, these networks still need the expertise of human element in the form network administrators. But the rapidly deployed networks used in disaster management face a very rare set of challenges. They are:

1. Lack of trained technical personnel on site to manage IT infrastructure failure.
2. The remote site network setups are very ad-hoc in nature and done in a short span of time.
3. Certain failures in connectivity and data flow can never be recognized as everything would look perfect from the outside.

Thus these new challenges should be addressed in a new way. The *National Academics* report on "**Improving Disaster Management: The Role of IT in Mitigation, Preparedness, Response, and Recovery" [7]** says that both agility and robustness of

`

the IT infrastructure is very crucial for the proper operation of the disaster management systems. Although it is a great idea to have a very robust and reliable system which is operational and keeps information floating all the while, from a more technical perspective it is important to understand that the entire IT infrastructure is volatile and prone to failure. Thus we need a solution which makes the disaster management system more agile and responsive to failures.

So with these factors in mind, we designed a reactive solution to address the vulnerability of disaster management systems: DINAR, Diagnosis of Network and Application Resources using a web based model. It is a network management methodology adapted to suite the nature of disaster management and similar systems.

Chapter 2 describes the Backgroud and Related Work. In chapter 3 we introduce DiNAR and talk about its architecture and design. This is followed by implementation details in Chapter 4. Chapter 5 presents the System evaluation and results.

`

CHAPTER 2

BACKGROUND AND RELATED WORK

Network resource management has been a traditional research problem. Over the years multiple solutions have been proposed and implemented. Simple Network Management Protocol (SNMP) [3] has been predominantly used by many commercial products. SNMP uses a Manager-Agent model, where every managed node in the network hosts a SNMP agent which reports health information to the SNMP manager. The architecture of an SNMP based management system is as shown in Figure. 2.

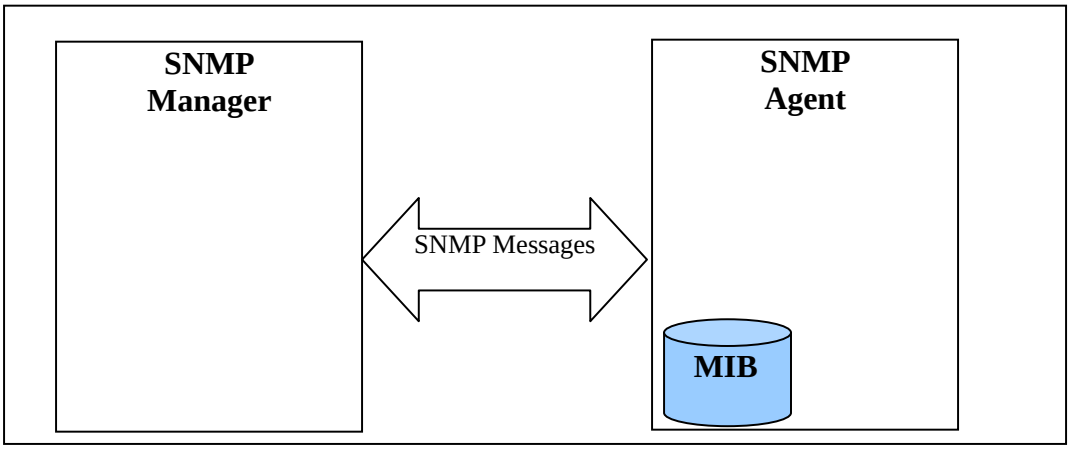

Figure 2 SNMP Architecture

SNMP Manager is a centralized node which is responsible for collecting the network information from each individual node which is running an ***SNMP Agent.*** The agent is software running on every node which collects health information about the local hardware and software environment. This information is stored in a localized database known as Management Information Base (***MIB***). SNMP assumes the entire manageable network to be under one administrative domain.

`

Outside SNMP, there have been lots of efforts in designing network management systems for different application scenarios. We will briefly explore three such designs.

1. Web Based Enterprise Management (WBEM):

WBEM [6] brings interoperability among management solution providers. It was introduced by Distributed Management Task Force to standardize the XML based network management protocols. XML based methods are a new class of management standards which use XML for data representation. WBEM defines a set of management and Internet standards to bring together management of distributed environments. WBEM is defined by three main components:

- Common Information Model (CIM): CIM is a standard which defines how to model network, application and other business processes in enterprise and service provider environments. It uses a standard object oriented structure using classes, properties, methods and relations (also known as associations).
- HttpAccess: Http acts as the transport mechanism in WBEM. HttpAccess component defines the specifics of http requests used to perform the CIM operations over the network.
- XmlCIM: WBEM defines *xmlCIM* which defines the xml grammar for mapping the CIM classes into XML elements and the CIM class properties into attributes. This is done by defining the XML Document Type Data (DTD). The DTDs specify the XML grammar for CIM. This component holds a very important key in achieving Interoperability.

`

With use of XML and HTTP WBEM has overcome one of the limitations posed by SNMP, i.e. the management traffic will not be blocked by service providers as it looks same as a web traffic. But the manager would still need the exact IP or URL to access the devices. This cannot be expected in the typical emergency response setup. This inspires for the design of a system management method which suits this special scenario where different entities are scattered across the Internet.

2. Ad-hoc Network Management Protocol (ANMP) [14]:

ANMP is a management framework with special design considerations for ad-hoc networks, mainly used in battlefield and emergency response systems [14]. It uses a policy based management mechanism. The network requirements are expressed as high level policies. There is a hierarchy of policy agents which realize these policies and also report management information to a global policy agent. ANMP was developed with two basic motives:

- It should be lightweight and suitable for Ad hoc networks.
- It should be compatible with SNMP.

ANMP uses its version of MIB to store the information of the Ad hoc network devices. It also supports alarms like SNMP traps to have asynchronous reporting of problems.

3. Yelp Announce Protocol (YAP):

YAP is a network configuration management scheme [15] which collects configuration settings from all the managed entities and stores them. The YAP architecture consists of a *YAP Server* and multiple *YAP Relays.* YAP Relays are like SNMP agents, but with a

`

difference. Instead of responding to requests from the manager, they relay the management information at regular intervals to the YAP server. The YAP server on the other hand collects as well as distributes configuration information through the relays. This theme of relaying data from the YAP Agents (Relays) without being polled by the manager, presents an interesting idea and food for thought. This scheme, although very pertinent to collection of configuration information, can also be applied to network health information collection.

As discussed earlier, emergency response systems and applications like Diorama pose a special set of challenges on the management methodology. This makes choosing of one among the above said protocols directly difficult. The primary reasons for this are:

- Non existence of a single administrative domain: All management solutions expect all the nodes to be within the same administrative domain. However, in emergency response networks, infrastructure elements are in separate sites and are connected through the Internet. Thus having one administrative domain is ruled out.
- Explicit addressing: Explicit addressing of managed nodes is a must. However, we can not provide such guarantee for a volatile and mobile network like the disaster management network.

In the next section we introduce DiNAR management solution which overcomes these challenges while using some of the principles used in all three of the modern network management methods presented above. DiNAR has been developed to adapt to applications of different needs. In this thesis work, Diorama has been used as a case study for implementing the proof of DiNAR concept. In the rest of the document, we use Diorama as an example under all scenarios.

`

CHAPTER 3

DINAR: CONCEPT, DESIGN AND IMPLEMENTATION

We understood in the previous chapters that present day network management methods altough effective and robust, are not suitable for applications which have infrastructure elements scattered across internet and rely on Internet as a backbone. It should be noted that we are not considering multiple office sites connected via VPN (over internet) as an example of such an application. Figure 1 shows a typical example of the target application. Ex. Diorama [Appendix A]. The primary reasons why SNMP, WBEM and other existing architectures cannot be used for applications like DIORAMA are:

- The communication between the manager and the agents goes through networks which are outside the administrative controls.
- Lack of explicit addressing. There is no direct way the manager can contact the agents and vice-versa using the addressing scheme (IP).

To overcome these challenges, DiNAR has been designed with the following design goals:

- Provide seamless information flow between agents and managers irrespective of the locations of the agents (provided there is network connectivity)
- Use a Push paradigm instead of a request response paradigm to overcome the need of explicit addressing.
- Finally to have the system as simple as possible at the disaster site. The aim is to avoid any configuration or setting up processes on the disaster site by the paramedics.

`

3.1. Dinar Architecture

DiNAR architecture follows a Manager-Agent based model and draws inspiration from WBEM[6] and YAP[14] in the way management information is represented and collected. DiNAR architecture consists of two main entities, the DiNAR *manager* and *agent*. An agent is a daemon service installed on all managed nodes in the field like PDAs, laptops, wireless routers etc. While a Manager which resides on a server in the remote control center is a central application to which agents report the management information. Figure 3 depicts the DiNAR architecture over the Diorama setup shown in Figure 1.

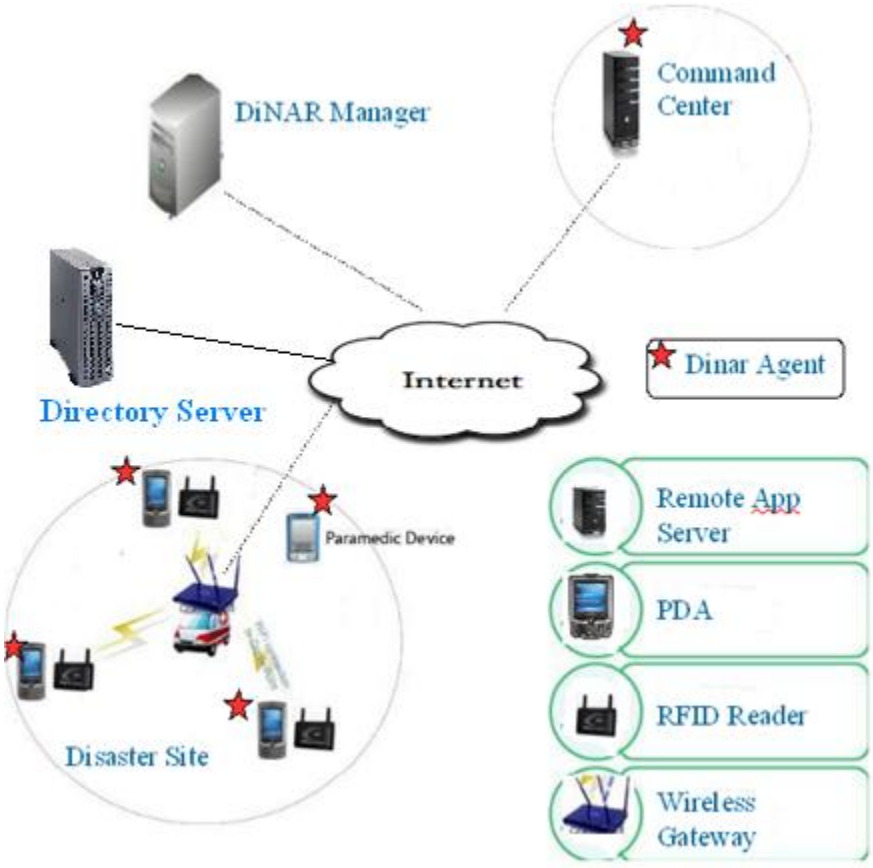

Figure 3.DiNAR Architecture

`

As shown in Figure 3, DiNAR consists of the following three major components:

1) DiNAR Manager: Manager is the centralized web server which collects information about all the nodes in the infrastructure. A detailed architecture of the manager and its subcomponents is discussed later in the chapter.

2) DiNAR Agent: Agent is a daemon process running on every computing node in the system (Remote clients, Web servers etc). Its job is to collect the health information of its environment and report it to the Manager. More details about the agent are discussed later in the chapter.

3) Directory Server: The directory server helps the agents to locate the manager on the World Wide Web. This acts a single point of reference to manager location and leaves the manager location flexible. More details about the directory server are discussed later in the chapter.

We now begin to analyze each component of DiNAR in greater detail and how they help in achieving the design goals of DiNAR.

3.2. Dinar Agent

The DINAR agent is a process which runs on every managed device. Its primary tasks are:

- Collect status parameters for the device on which it is running and also from interfaced gadgets and applications running on the same machine.
- Contact and establish connection with the DINAR Manager.
- Send updates at regular pre decided intervals.

`

The agent is an independent application running on the device and has no relation with the other applications that are running. Figure 4 shows a block diagram view of an agent. The functionality of an agent has been split into multiple components with a motive to keep things simple and modularized. It also allows future expansion and customization of the agent. Let us look into each component in detail:

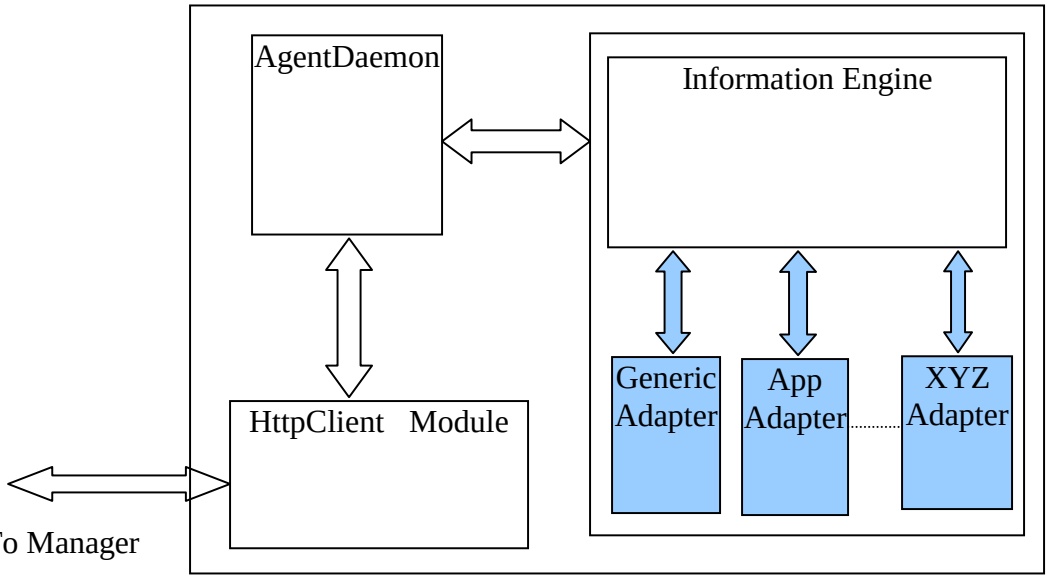

Figure 4 DiNAR Agent architecture

- **Agent Daemon:** This is the main backbone process which controls all the other components of an agent. Whenever the agent is started, it is this daemon process which kicks off. One of the initial tasks for this daemon is to locate the manager and initiate a contact. Locating the manager is achieved using the Directory server which is described in greater detail. So as of now we can safely assume that the manager location (URL) is known. After the initial contact with the manager, the agent starts the information engine and receives regular updates from it. This update information is structured and passed onto the manager at regular intervals. Thus agent daemon is

`

the nucleus of the DINAR agent and controls all its operation.

- **HttpClient module:** This is the communication engine of the agent. The agent communicates with the manager using the Http protocol. The HttpClient module manages the establishment and maintenance of the Http Connection with the manager. It uses the Apache HttpClient 3.1 library to perform all the Http operations. It also tries to maintain only one connection to the manager throughout and keep this connection persistent. This means all Http messages will be sent on the same TCP connection. Persistence is achieved using Http 1.1 which pipelines multiple requests onto same HTTP connection. DiNAR manager expects the connection from the agent to last until it's operational in the field under ideal circumstances. The manager sets a timeout value of 60 seconds. This means any agent idle beyond 60 seconds is considered to be unreachable for the moment. Although this does not brand the Agent or the link to be down. The manager does further processing to determine if agent is truly down. More details about event processing done by the manager are described in Section 3.3.4 (Analysis Engine).
- **Information Engine:** The information engine is like the Management Information Base (MIB) in SNMP. It stores all the relevant and needed status parameters and tracks them continuously. As described above, the agent daemon interfaces with the information engine to get hold of the current state of the machine and pass the information. The information engine in itself is of not much value. It just acts a central docking point for all information adapters. Information Adapters are specialized modules for continuously monitoring a certain environment and getting its state information. In Figure 4, all modules displayed in blue represent

`

information adapters. They are started in cascade to the Information engine. Individual adapters are invoked by the Information Engine at a predefined interval to collect the present state information for the components they are responsible. E.g. an adapter responsible for collecting the health information of all the aplications on the Diorama server is invoked every x seconds where x is configurable based on the user's requirement. "x" here is called the refresh interval. DiNAR sets the default value of 60 seconds. This is arrived at based on the tests conducted on the performance of the Agent process for varying refresh interval. We then did a trade off between the performance and the freshness of the information to conclude 60 seconds as an ideal refresh interval.

- **Information Adapters:** As mentioned above information adapters are specialized modules for continuously monitoring a certain environment and getting its state information. The number and class of adapters can be customized. If the device is a hand-held device in the field, then it can have adapters for the interfaced gadgets, applications etc. If the device is a central web server, then it would mainly contain application adapters. The application adapters keep track of all the managed applications and monitor their status continuously. We developed an application adapter for Servers and other Client machines running windows. This gives us the dynamic availability and performance statistics of the listed applications. Apart from the specific adapters, all agents have a mandatory generic adapter. The function of generic adapter is to gather vital parameters of the health of the machine itself like CPU usage, memory usage etc. Except the generic adapter, the rest are optional. This architectural decision on information adapters will help in making DINAR

`

apply to wide variety of scenarios. Anybody can write a custom adapter and plug it into the agent. This can help a great deal in customizing the agent according to the needs. We developed the generic adapter and an application adapter for windows environment.

3.2.1. XML Representation

As discussed above, XML is used to model the management information. In this section we will take a look at sample XML update sent from an agent and understand its structure:

As shown above, every agent update begins with a “DINAR” start tag. This is followed by the tag which represents the type of device which is reporting. Within this we have the XML tags representing the components which the agent is collecting data for. In

`

some cases there can be meta tags like the one shown above e.g. *DINAR_GenericAppHeader.* The meta tag is used by adapters to group multiple peer tags coming from same adapter.

3.2.3. Agent ID

AgentId is the unique id to represent the agent in the manager repository. To ensure uniqueness across multiple sites, we use the MAC address of the device to be its AgentID. This helps maintaining the uniqueness off the agent.

3.2.2. Agent Bootup and Configuration:

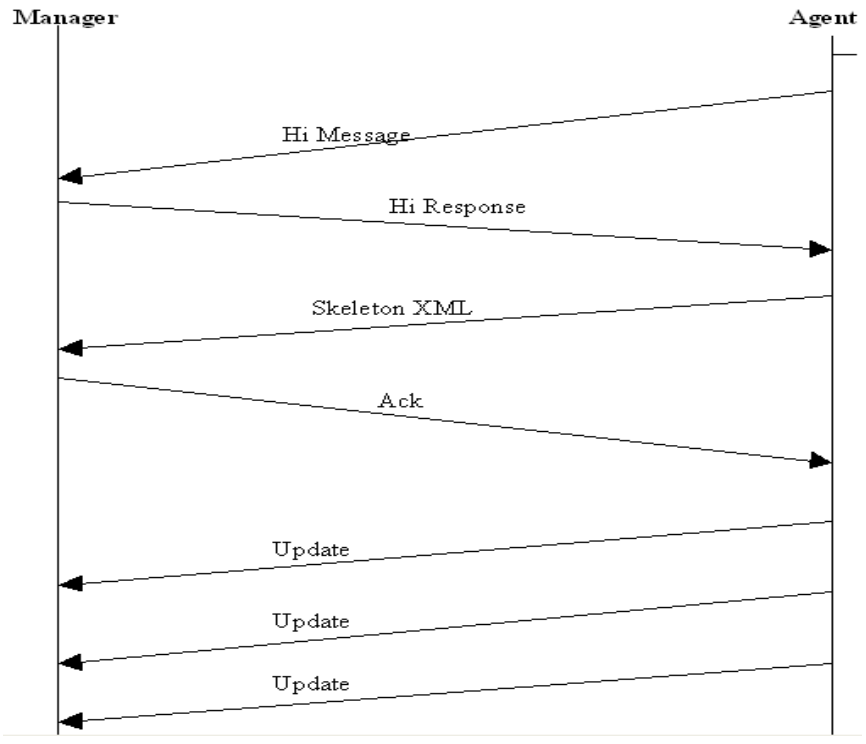

Figure 5.Manager Agent interaction

`

As soon as the agent boots up and knows the URL at which the manager is residing, it first issues an HTTP request with *messageType* set to “Hi”. Along with this, it also sends a polling interval, which indicates to the manager the frequency with which it should expect updates. If the manager receives all this information correctly, then it would send a “Hi” response. The agent which receives the “Hi” response now knows that the initial handshake has ended succesfully. The agent now sends the skeleton XML document which represents the device, applications and the attached gadgets which the agent will be monitoring. There are no status updates in this XML document, just the skeletal XML. After receiving this skeletal document successfully the manager sends back an ACK to indicate to the agent that everything was received successfully. After the ACK is received, the agent starts a timer and after every t secs (equal to polling interval) it sends out the update XML document. Figure 5 depicts these handshake messages. This continues unless and until the agent or the manager are stopped or the communication is affected.

3.3. Dinar Manager

DiNAR manager is the focal point of the whole DiNAR system. It is the information sink for all the DiNAR agents running across the environment. Hence we can define the DiNAR manager as a central application which receives and processes the management information. The manager can also be visualized as a web server listening for agent updates on one of the Http ports. The Figure.6 shows the block diagram view of the DINAR Manager and its components.

- **Manager Servlet:** This module acts as a global interface of the manager. It listens

`

on the assigned manager port and accepts the updates from all the agents. The updates from the agents are in XML format. The servlet extracts the xml data and passes it to the XML parser for further processing.

- **XML Parser:** The XML parser module parses the incoming xml data from the servlet and creates objects for each of the Element in the xml document. DiNAR manager uses a DOM parser. Each Incoming XML tag is converted into an object (except few meta tags) defined in the Data model (Explained in section 3.3.2).
- **Object Pool:** The DINAR manager looks at every device and its components and attachments as an object defined in its Data model. Even the DINAR agent is represented as an object in the pool. It instantiates an object for every new device or component it manages. This object is an abstraction of the real device. Whenever updates are received for an already created object, only the property of the object is updated. Objects follow the class structure defined in the DiNAR data model which represents the blueprint of the topology and all its classes.
    - Alarm Module: This module is responsible for triggering alarms based on the properties of the objects in the object repositories. The agents transmit abnormal activity reports using the *status* attribute for each object. Based on this, the alarm module triggers alarms to highlight the problem. These alarms are then picked up by the analysis engine for alarm correlation.
- **Analysis Engine:** The goal of this module is to analyze the management data received from the agents. It consists of an event correlation engine which correlates the alarms. The Analysis engine is explained in greater detail in section 3.3.4**.**

`

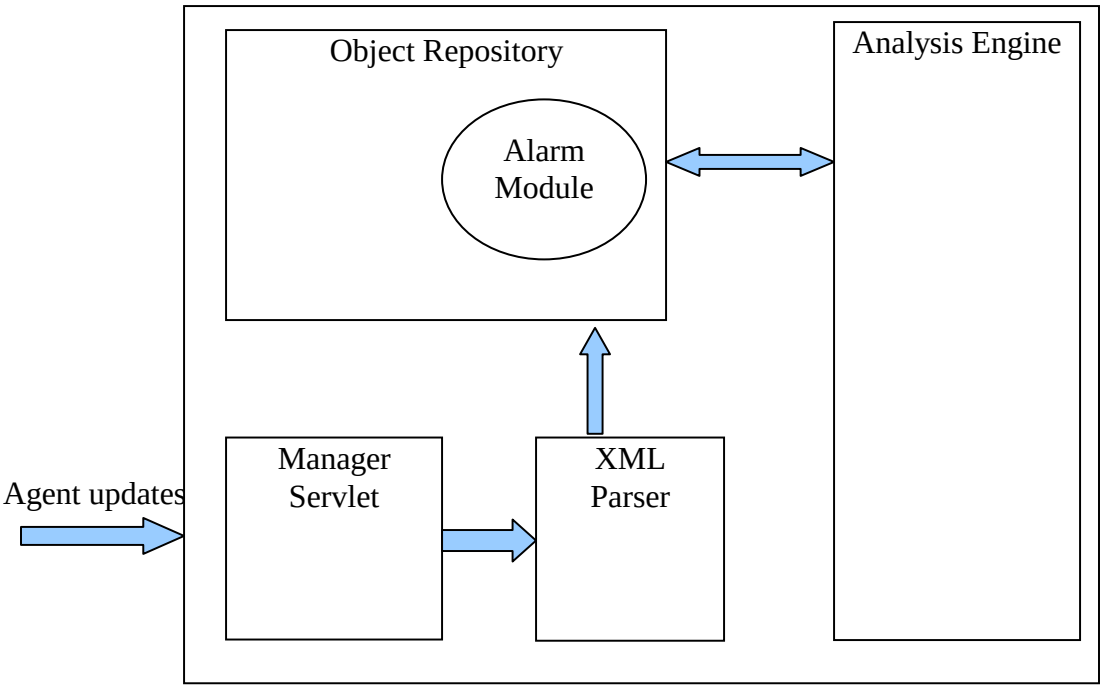

Figure 6.DINAR Manager Architecture

3.3.1. Alarm Model

Alarm model defines the set of alarms that can be triggered by the *Alarm Module.* It is very specific to the application for which DiNAR is being used. We developed an alarm model for DIORAMA[Appendix A] application. Figure 7 shows the alarm model for DIORAMA application.

The alarm model is implemented as an inheritance of Java classes where each event inherits from a parent event. In Figure 7, *Event* is the base alarm which contains basic properties of an alarm. This is then inherited by the other other alarms as shown in the Figure 7.

`

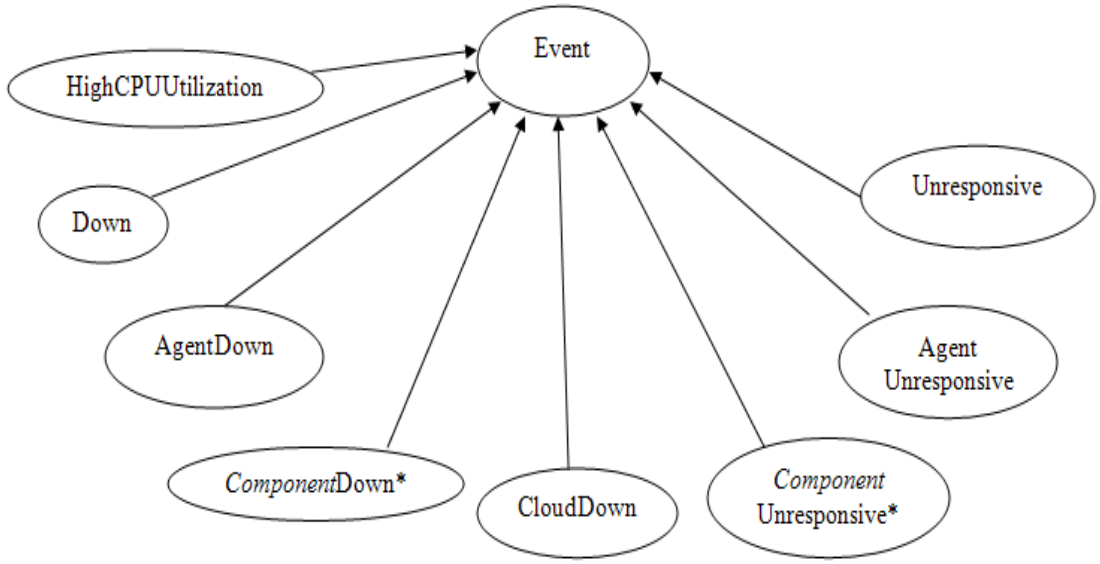

Figure 7.DIORAMA Alarm Model

3.3.2 Data Model

DiNAR Data Model is the blueprint which the DiNAR manager uses in creating the abstract topology. The is represented by an interrelated set of classes which are specific to an application. In this section we will demonstrate the modeling of DIORAMA system. Data model forms an important part of the application as all the objects in the object pool of the manager depend on the data model. For the purpose of DIORAMA, we designed a data model encompassing all the components of DIORAMA. Figure 8 below shows the complete data model for DIORAMA. Each of the class is represented using a rectangular box. For example *DINAR_ClientDevice* is a class which represents any generic computing device in the disaster site. These classes are then linked using arrows. These arrows indicate class inheritance. Lets consider the DINAR_*ClientDevice* example. This class has two subclasses: *DINAR_TrackDevice and DINAR_SiteDevice.* The former represents a D-Track device as defined by DIORAMA [Appendix A] while

`

the later represents a generic purpose remote client. The arrows used always point towards the base class. Other than the directed arrows, the other connections between the classes are known as relationships between them. Relationships are all unidirectional and each relationship has a converse to it.

The naming of the classes and the relations has been done following the CIM guidelines. CIM classes are named in two strings separated using an "_". The first string is known as the domain name and should be constant throughout the model. This model all in all represents the entire DIORAMA system. Figure 8 shows the diorama data model. In this model, even DINAR_Agent is treated as an object and as a part of the repository.

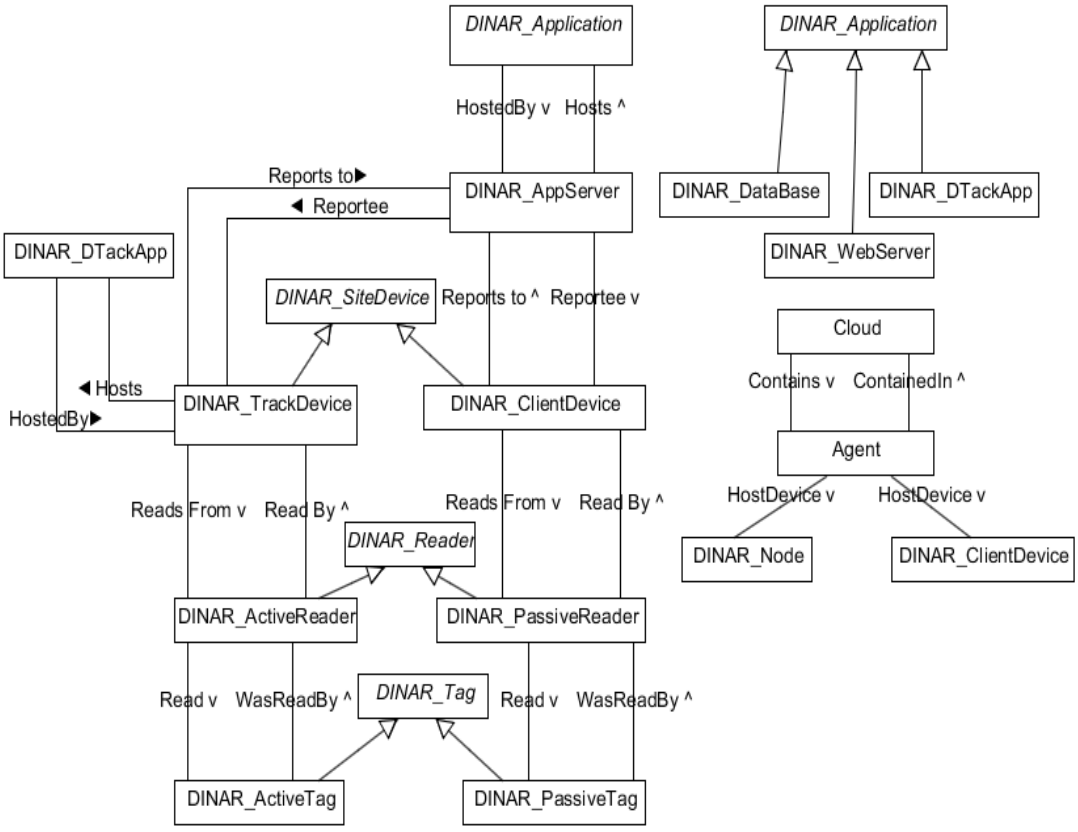

Figure 8.DiNAR Data Model for DIORAMA

`

3.3.3. Cloud Aggregation

DiNAR was aimed to be applied on distributed applications which have multiple sites and whose elements interact over the internet E.g DIORAMA [Appendix A]. In DIORAMA, the disaster site devices connect to the internet using a wireless LAN (WLAN). This wireless LAN comprises of a wireless router. To model this network scenario, we defined a new entity in the DiNAR model known as “Cloud”. A cloud is meant to represent a particular disaster site location. Every agent sending updates from a particular disaster site is believed to behind a cloud representing its site. Figure 9 shows a capture from the DiNAR console where two agents reporting from the same site are clubbed under a single cloud.

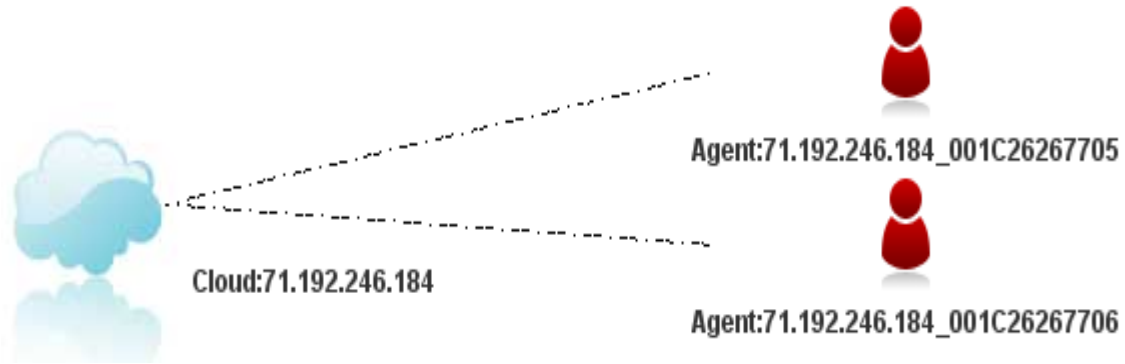

Figure 9.Cloud

The source IP of the HTTP connection coming into the manager is used to define a cloud. All agents within a single WLAN will be having the same source public IP.

3.3.4 Analysis Engine

The analysis engine is the module responsible for performing alarm correlation on all the triggered alarms in the DiNAR repository. Figure 10 shows a block diagram view of

`

the analysis engine and its components.

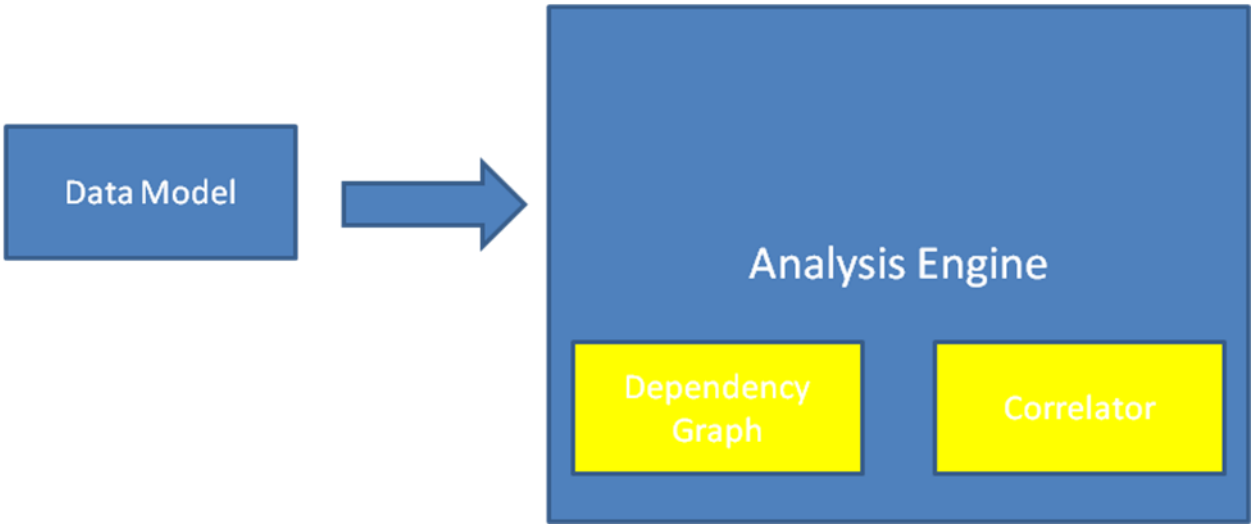

Figure 10.Analysis Engine

- **Dependency Graph:** Our Analysis engine has been built using the Dependency graph based event correlation algorithm [8]. Dependency graph is a mechanism where the different classes in the data model share a dependency relationship between them. This dependency can be read as "Fault in class A will lead to Fault in class B" [8]. Thus class B is dependent on A. This schema helps us to build a hierarchical structure of dependencies among components. The alarm correlation algorithm uses the dependency graph as an input. To build the dependency graph we use the DiNAR Data model as a reference for relationships between the devices. While designing the dependency graph we condensed the graph as much as possible to its base classes. Thus if any class in the Data Model is not shown in the dependency graph, its base class needs to be looked up. Figure 11 below shows the Dependency graph for DIORAMA.

`

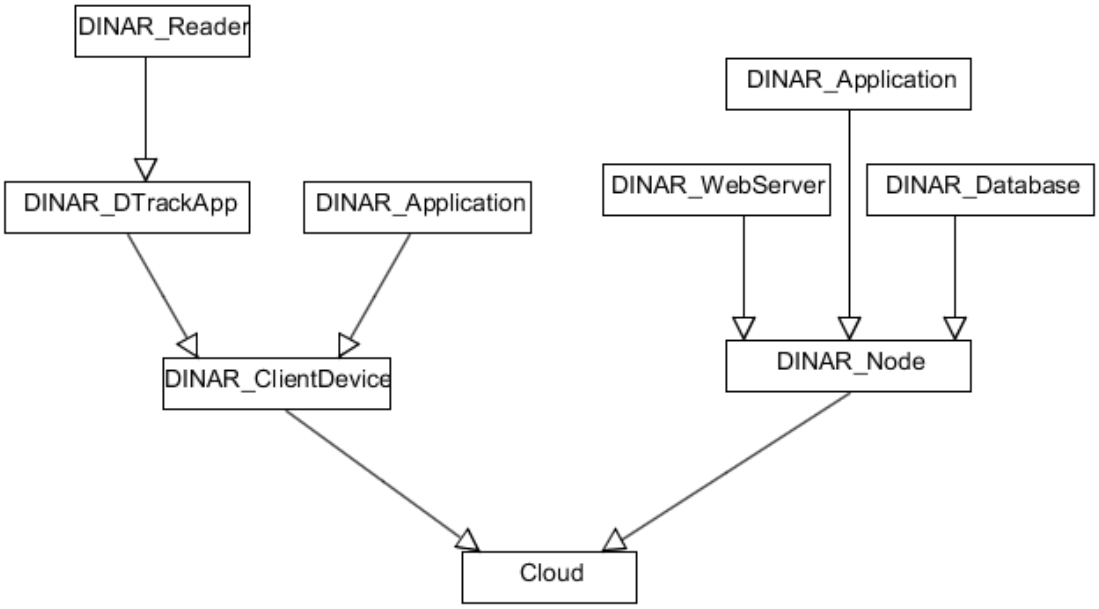

Figure 11.Dependency Graph

- **Correlator:** The correlation algorithm uses the dependency graph to reach to the root cause of a fault. It correlates multiple alarms and filters out alarms which are caused due to failure in parent components. It also helps in pointing out the more pertinent problem, resolution of which can be a possible resolution of the other. Although this might not be always true. The dependency graph correlation algorithm [8] implemented in DiNAR works as explained below
    - The correlator kicks in every 60 seconds and builds the dependency graph using the above blueprint.
    - It then traces through the graph and assigns respective alarms to each object (if any).
    - It then marks all the leaf nodes with alarms and starts analysing them in a loop

`

- For every alarm in a node, it checks to see if the object on which it depends has an alarm. We define a certain set of alarms to be unrelated. E.g a HighCPUUtilization alarm on the Application server is not related to an ApplicationDown alarm although Application Classes are dependent on the DINAR_Node class. If the alarms in the parent and the child object are not *Unrelated,* then the alarm in the parent is considered to be the cause of the algorithm in the child object.
- If the alarm in the parent object is concluded as the cause then the alarm in child object is marked as a symptom and the parent object alarm is now processed recurcively. If not, the child object alarm is itself marked as a root cause.

3.4. Directory Server

Locating a manager involves finding the exact hostname (or IP) and the http port on which it is listening. Launching a full fledged directory service would mean, the agents having the capability of finding the manager address using broadcast. However this is not feasible in the present setup. So we add a step of indirection to reach to the actual manager. Instead of the manager's address, all the agents would contain the address of a pre coded directory server. This directory server will then know the address to the actual manager. So as soon as the agents boot up, they will first contact the directory server and request for the location of the manager. The directory server responds with the manager URL. Hence with one additional step, the agent can get the manager address dynamically. Even if the manager location changes, only the directory server needs to be

`

updated and not all the agents. Thus, this approach is more scalable. The requirements of a directory server are to have a URL which is constant all the time.

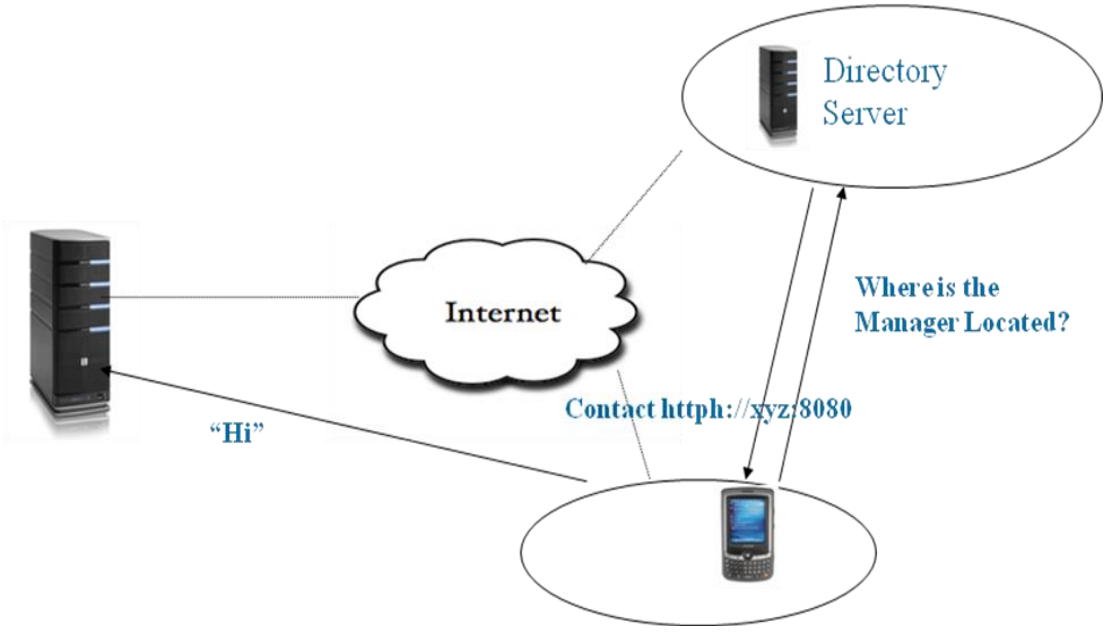

Figure 12.Directory Server Operation

`

CHAPTER 4

IMPLEMENTATION DETAILS

4.1 Dinar Agent:

This has been implemented in core Java SE. Some of the highlights of the agent development were:

- It is a single threaded application. This was done mainly due to
    - Reduce load on the agent machines
    - Agent has a fixed update interval. Hence data collected between update cycles is of not much use if overwritten
- Uses HttpClient 4.0 module to perform http 1.1 operations.
- Agent was developed only on Java Standard Edition. Hence it cannot be ported to a mobile device. Attempts to port the agent onto a mobile device were not succesful owing to the stripped down version of the Java Mobile Edition which does not support the core HttpClient 4.0 API. Hence development of agent on mobile devices needs to be done using the local development frameworks provided by the vendor.
- It uses the NSClient service to pull the health information from Windows hosts like *uptime, cpuUtilization.*

4.2 Dinar Manager

DiNAR manager is a multithreaded J2EE application. It contains of two main threads:

- Thread 1 : Main thread responsible for collecting updates from agents and creating the abstract topology

`

- Thread 2 : Alarm correlation engine thread

Manager uses the XML Schema Definition (XSD) to validate the incoming data before processing the agent updates.

`

CHAPTER 5

SYSTEM EVALUATION

The previous chapter described the details of DiNAR system architecture and implementation. To summarize, DiNAR system can be broadly visualized as shown by Figure 13.

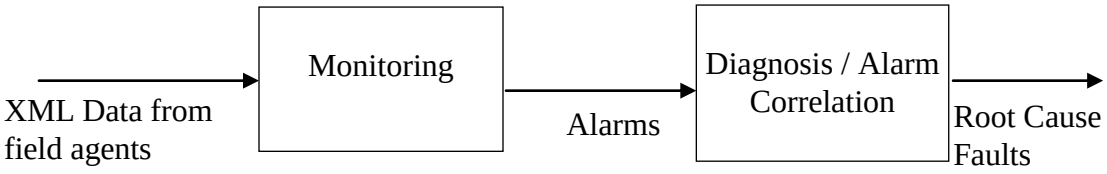

Figure 13.DiNAR Summarized

The system evaluation strategy was concentrated on segregating every independent component of the system and testing it individually. It was the followed by a complete feature testing to test the resultant output of the system. This chapter discusses the testing schemes, parameters considered and the test bench details followed by the results of the system evaluation.

5.1. Testing Scheme:

DiNAR system evaluation tests have been classified as:

- Tests in the Monitoring phase.
- Tests in the Diagnosis phase.

Figure 14 below shows the DiNAR test model and the functionality being tested at the end of every phase (sub phase). The system evaluation used two testing methodology.

1) Live topology testing.

2) Simulated topology using test bench.

`

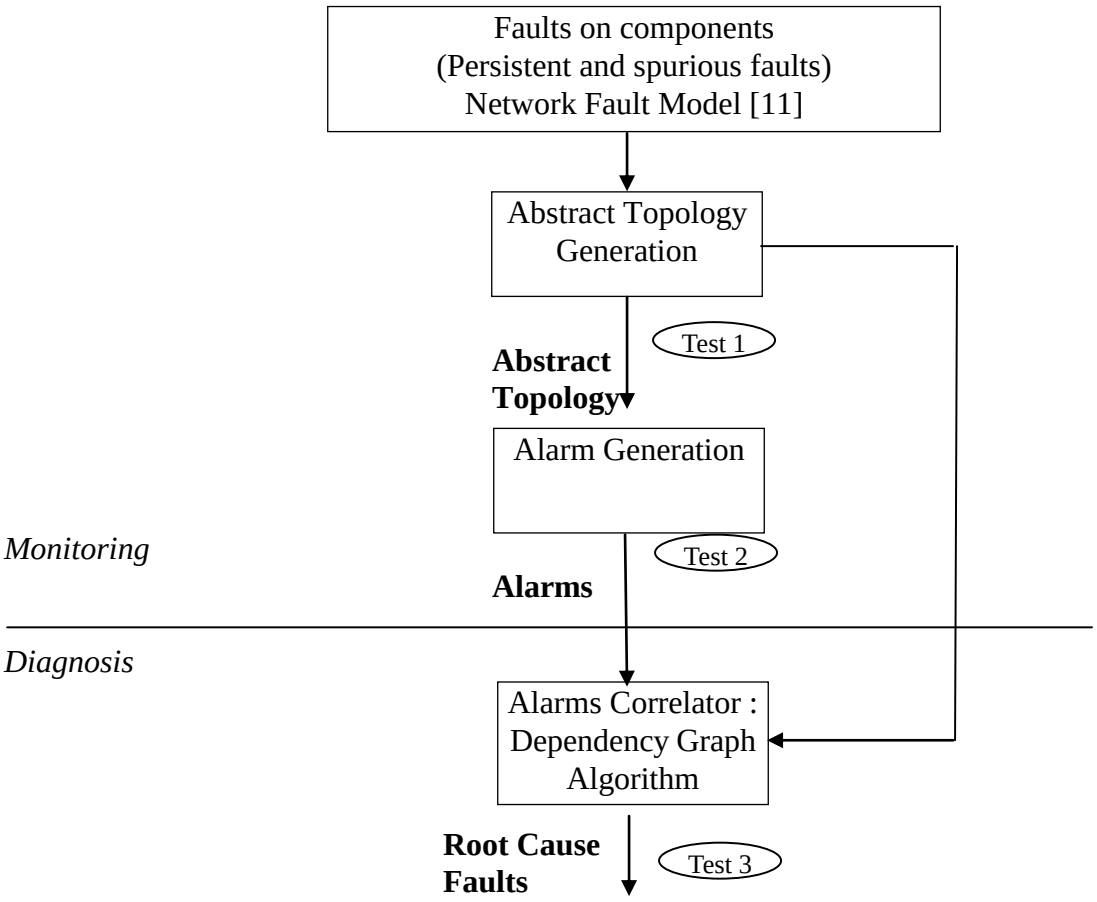

Figure 14.Testing Model

1) Live topology testing: DiNAR agents were deployed in three laptops running Windows operating system and on a web server running a server version of Windows. One of the laptop agents was made to portray as a mobile device agent. The web server agent was in a separate cloud thean the rest. The manager collected health information from all the agents. Faults were induced into this setup. We used the Network Fault Model (NFM) [11] which defines 5 different types of faults possible in a networked application.

- Drop

`

- Cut
- Lost
- Corrupt
- Carrier

NFM faults are designed for individual packets, but as DiNAR abstracts packet level details, we considered every HTTP request/communication as an atomic entity and applied this model.

2) Simulated topology using test bench.

We developed a test bench to simulate multiple agents. Each agent is feeded with a separate configuration file and an internal topology. The agents in the test bench function like any normal agents, except that it does not have an Information Engine. It picks up the topology information from a hard coded XML. The agentID of the simulated agents is preassigned using a configuration file.

The test bench introduced the faults as mentioned in the Network Fault model [11] using a *FaultInduction* thread. This thread is a daemon which introduces faults by.

- Stopping and starting agents.
- Changing status information of individual components.

5.2. Test Cases and Results

Figure 14 shows three tests (Test1, Test2 and Test3) conducted at various breakpoints in the DiNAR system. Apart from these, we conducted two more tests to understand the performance of the system and to obtain optimal update and refresh time intervals. In this section we look and analyze the results of the tests.

`

5.2.1. Test 1: Abstraction Accuracy

Abstraction accuracy is the measure of closeness between the actual topology in the field to the abstracted topology in DiNAR manager. We define this metric based on the relationships between each component. To measure this subjective metric, we used the following technique:

- On the agent side,
    - (a1) Every component (sub component) to be measured is assigned 1 point.
    - (a2) Every relationship to be built is assigned 1 point.
- On the Manager side
    - (m1) Every component seen on the manager is assigned one point.
    - (m2) Every correctly built relationship between the components by the manager is assigned one point.
- (a3 and m3) Finally, the correct grouping of agents under respective clouds (reflecting their locations physically) needs to be considered. To correctly capture this metric, we assign 1 point for every cloud object seen on the manager (a3) and 1 point for every remote site under consideration(m3).

All the points from both the agent side and the manager side are summed up. The *Abstraction Accuracy* is finally calculated as

$$(m1 + m2 + a3) * 100 / (a1 + a2 + m3)$$

Note that m3 and a3 are swapped to give a correct meaning. Theoretically it is not possible to get $m3 < a3$.

`

We conducted tests using the live physical topology as mentioned in section 4.1 and simulated agents using the DiNAR test bench with varying number of agents (1-6) under each site. The topology consisted of two sites to depict the disaster site and the command center.

Results:

The tests were conducted considering the following assumption. All agents in a single site were connected to Internet through a wireless LAN which had a single vertical connection. This meant all of them were behind a natted gateway. Under these circumstances we observed 100% accuracy in abstraction.

The other network setup which should ideally be considered is when every agent device has a separate vertical link. This is scenario when every client device in the disaster site has 3G connectivity and the site has no internal LAN. From DiNAR perspective, the topology remains the same per agent. The only factor that gets affected is the classification of agents into respective Clouds. This depends on the external IP of the device individually unlike the case of a LAN network. The external IP is allotted by the base station to these 3G clients using their Access Point Names (APNs) and is not consistent for all the devices. We could not completely test this network scneario due to lack of enough hardware for 3G/GPRS connectivity. But we observed the assignment of external IPs using 4 iPhones with 3g connectivity. It was observed that irrespective of same location, all of them got separate external IPs all the time. This would mean DiNAR manager will show all 4 of them in separate clouds.

Thus assuming we had hypothetical agents installed on each of these iPhone, the abstraction accuracy of this scenario would be $(x)/(x + 3)$.

`

Where x is the sum of points for all other components.

5.2.2. Test 2: Spurious alarm suppression

One of the biggest challenges faced by Network management systems is the ability to handle spurious alarms. Spurious alarms are unavoidable in any network scenario. Such alarms originate mainly due to:

- Temporary loss of connectivity
- Temporary unresponsiveness of applications.
- Etc.

DiNAR uses a wait and hold approach to handle spurious. The wait time before an event is considered non spurious is equal to two update cycles. To test spurious alarm suppression and robustness of DiNAR we used the Network Fault Model (NFM) [11] which defines 5 types of faults possible in a packet based network. Although it's not completely appropriate in our scenario, we only consider a subset of this model. The fault types Drop, Lost, Corrupt are relevant in DiNAR perspective. The other two, Carrier and Cut are very specific to packet level granularity and are not relevant in a system which abstracts at the level of HTTP.

Test: The goal of this test is to check the robustness of DiNAR against errors and spurious alarms. The test bed was created using the DiNAR test bench. The topology consisted of 3 simulated agents per site and two such sites. The update interval was set to 60 seconds. The Network Fault Model was then applied by inducing the following faults:

- Drop: We induced this fault by introducing syntactical errors in the XML being sent. This leads to the server dropping this update.

`

- Loss: This error type was achieved by stopping the agent for a varied time intervals between 0 seconds to 180 seconds (at samples of 10 seconds) successively. The time instance at which the agent is stopped is also critical in this test. For this we chose the two extreme ends. The agent was paused once immediately after a previous update. We called this as t=0. The next test had the agent stopped right before an update. We called this as t=60.
- Corrupt: Corrupted data can be of many forms. In our tests we considered corrupted data in two ways.
    - Wrong status information about components. Involved flipping of the status from *Running* to *Crashed* etc. We flipped the status information of components. 5 such components retained this curruption for a single update cycle, while remaining 5 components retained this corrupted status information for 2 or 3 update cycles.
    - Syntactically correct, yet semantically wrong information which does not mean anything from DiNAR perspective. This test involved changing the "status" attribute of a component to "XYZ" and keeping it same for 10 consecutive update cycles.

Results:

- **Drop:** All the syntactically errorenous XML messages were rejected without any failure. But as the HTTP connection is still alive, an Unresponsive alarm for the device is not fired and this was a just result. Yet this was logged in the server logs but not shown in the GUI as a design consideration.

`

- **Loss:** We tabulated the percentage of spurious alarms suppressed under both the cases and is shown in Table 1 below. The suppression window shows the minimum amount of time for which spurious alarms can be suppressed for two different values of "t".

Time Instance (t)	Suppression Window (in sec)
0	180
60	120

Table 1 Spurious alarm Suppression Window

- **Corrupt:** For the two types of data corruption mentioned above:
    - Wrong status values:
        - For all the components which held the corrupted data for just 1 update cycle and then the corrected values were injected, DiNAR manager was successful in suppressing the alarms as transitive and no alarm was generated.
        - For the components which transmitted the corrupted data for 2 or 3 update cycles and then cleared the alarm, alarms was generated which eventually were cleared. However, these were shown on the GUI and were a part of the Alarm Correlation process.
    - All incoherent values in the XML input from the agents were ignored by the DiNAR manager. Yet this was logged in the server logs but not shown in the GUI as a design consideration.

`

5.2.3. Test 3: Diagnosis Phase testing

In Chapter 3 we introduced the dependency graph event correlation technique to diagnose the alarms generated by the DiNAR agents. To measure the effectiveness of this method in our application, we used the following two metric.

- **Convergence time:** This metric indicates time taken for the dependency graph algorithm to reach to its final conclusion. It is measured from the time the algorithm is invoked with a set of alarms to the time it reaches its final conclusion.
- **Percentage False positives/False negatives of root cause detection:** This metric indicates the number of faults wrongly detected (or not detected) in the experimental setup.

Test Conducted:

To compute the convergence time of the correlation algorithm, we used the simulated test bench with 20 agents. The performance of the dependency graph based correlation algorithm depends on the depth of the graph (L) for every correlation [8]. Based on the dependency graph for DIORAMA, we have maximum depth of L=3. Our test cases consisted of generating 1-20 alarms of both L=2 and L=3 depth. We calculated the time taken by the correlation algorithm to reach to its end for each of the 20 X 2 cases. Figure 15 shows the plot of convergence time against number of alarms in the system.

One factor which affected the convergence time was the thread dispatching schedule of the JVM as correlation engine is a separate Java thread. However we reduced the effect of thread dispatch by having no I/O statements.

`

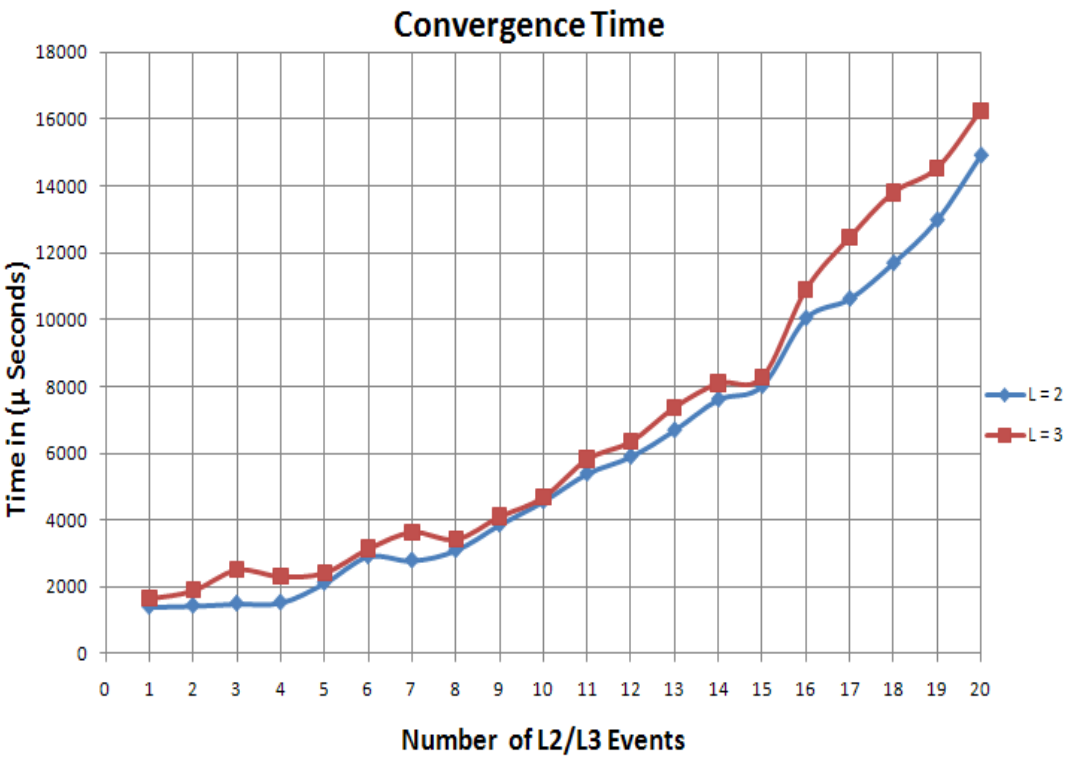

Figure 15.Convergence Time

Correlation accuracy and false positives: Event correlation in DiNAR unlike other network management applications is less complex. This is due to the unidirectional architecture of dependencies. Accuracy in correlation is a metric to measure the outcome of the event correlator. In the above experimental setup, following were the shortcomings observed in our event correlation engine.

1. A Device down alarm which was adjudged as the root cause overruled other actual alarms from its components (E.g. Application down) even though the later was a correct alarm and not a symptom of Device being down.
2. Cloud unresponsive alarm is based on the status of all the agents. It assumes that if all agents are down, then there is a network level problem in the site (Cloud).

`

Apart from these two, the correlation engine was successful in showcasing the correct and the most pertinent problem with highest amount of severity. Although point number 1 above discusses a behavior of the engine where actual alarms from subcomponents are suppressed by actual alarms from its dependent component, it does prove to be logical from administrative point of view. For example, in a case where we have an alarm for both the Device and its hosted application being down, it is logical to advise the administrator to first look at the Device failure problem. Table 2 shows a summary of the results of correlation for different failure scenarios.

Failure Scenario	**Correlation Algorithm Successful in pointing to all the right problems?**
Only ApplicationUnresponsive	Yes
Only ApplicationDown	Yes
Only HighCPUUtilization	Yes
Only RfidReaderDown	Yes
Application and Host device Down	Yes
RFiD Reader and Host device Down	Yes
RFiD Reader and Interfacing application Down	Yes
RFiD Reader, Host device and Interfacing Application Down	Not Always
Application/RFiD reader Down along with High CPU Utilization in the host device	Yes
All agents in cloud Down	Yes
Only 1 agent in the cloud is Up	Yes

Table 2 Summary of Correlation Results

`

5.3. Agent Overhead:

In this section we discuss the overhead posed by the Agent running on the devices. We consider two types of overhead here:

- Bandwidth Overhead
- CPU Overhead

5.3.1. Bandwidth Overhead

Bandwidth overhead is measured as the additional bits per second (bps) contributed by each agent. To measure this metric, the test setup consisted of four live agents connected to the internet through a wireless LAN. We used the Cradle Point setup as used in a real DIORAMA scenario. The test consisted a mock 10 min sessions involving all the 4 agents reporting health information about the device and monitoring 3 applications. For each 10 min sessions we varied the update interval ranging from 10 seconds to 120 seconds and measured the bandwidth addition (in terms of bits per second) by each agent. The bandwidth was measured for the entire http flow between the agent and the manager.

Figure 16 shows the per agent bandwidth contribution for different update intervals. It needs to be noted that, these values are specific to the amount of information being sent, which depends on the number of information adapters. If there is significant increase in the number of information adapters, then the bandwidth addition will be higher. However, the relative proportions will remain the same.

`

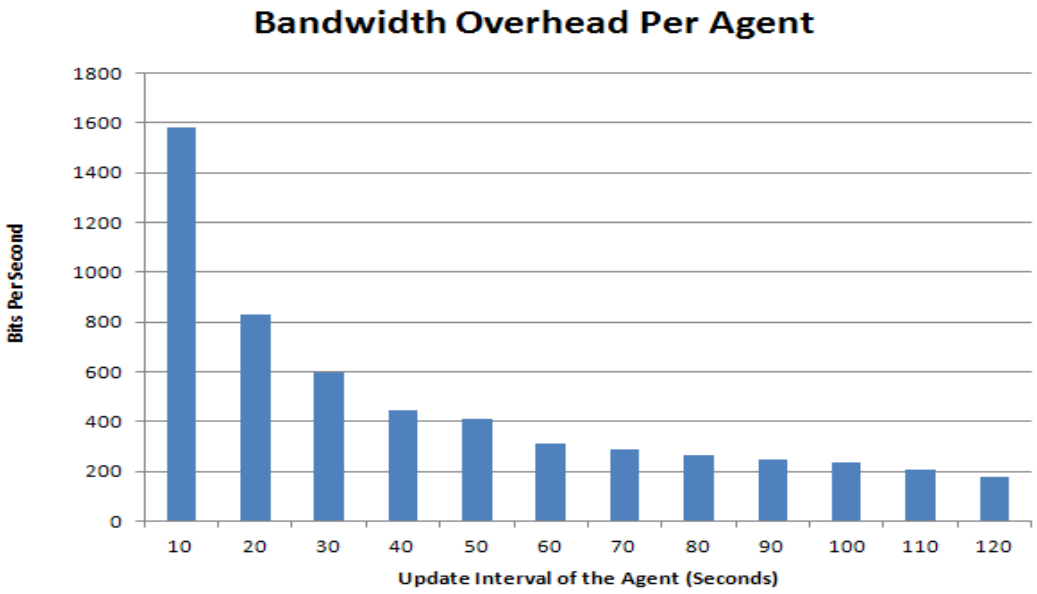

Figure 16 Bandwidth Overhead

5.3.2. CPU Overhead

CPU overhead is a measure of the processing overhead added by running the Agent on the agent devices. To measure the CPU over head we calculated the total time taken by one update cycle of the agent. This involves the time to collect information from all the adapters and transmit it across to the manger. For the test scenario mentioned in section 5.3.1, the average CPU time for one update cycle of the agent is 1224 ms.

For the mock 10 min test scenario, with varying update intervals, the total CPU time can be represented as 1224*U ms, where U is the number of update cycles. The lower the update interval, higher is the value of U and more is the CPU overhead.

5.3.3. Choice of Update Interval

The results obtained in section 5.3.1 and 5.3.2 helps us understand the choice of 60 seconds as the ideal update interval for all DiNAR agents. Figure 16 shows the varying bandwidth overhead for varying update intervals for a fixed duration of time. Based on

`

this, it is ideal to have longer update intervals which reduce the overall overhead of the agent. But longer update intervals lead to less fresh data at the manager. It also means slow realization of failures by the Manager. Table 3 displays the minimum time needed to fire the Device and Component alarms for different update intervals.

Update Interval	Device Alarms (Sec)	Component Alarms (Sec)
10	80	20
20	100	40
30	120	60
40	140	80
50	160	100
60	180	120
70	200	140
80	220	160
90	240	180
100	260	200
110	280	220
120	300	240

Table 3Alarm Generation Time

As seen in Table 3, the time needed to trigger an alarm after a failure grows significantly for higher update intervals. For update interval of 120 seconds, the time to obtain device failure alarm is 5 mins. This is a very long time period for a scenario like DIORAMA where the actual times of operations. At the other end of the table, the duration to trigger

`

device alarms are in the range of 80/100 seconds. The duration for component alarms is 20/40 seconds. This is a very less amount of time to trigger the alarm considering the transient failures. Such low time periods for alarm generation will also affect the windows size for spurious alarm suppression as discussed in section 5.2.2.

Thus the choice of update interval was made considering a tradeoff between alarm generation time and agent overhead. Hence update interval ranging between 50-70 seconds is a good design choice considering the trade offs.

`

CHAPTER 6

REPAIR AND RECOVERY MEASURES FOR DIORAMA

In chapters 3, 4 and 5 we presented the details of DiNAR system and its evaluation results. DiNAR as a tool helps guide the system administrator to have a complete view of the IT infrastructure elements of DIORAMA. It helps detect the faults and analyze these fault alarms through an alarm correlation engine. In this chapter we attempt to provide the corrective actions that can be invoked based the alarms raised by DiNAR.

To understand these corrective measures, we divide the whole DIORAMA setup into its two predominant zones

- Disaster site
- Remote site (Command Center)

6.1. Disaster Site

The disaster site is the area which generates the information about a particular disaster which has just occurred. This information being generated is very critical to the correct portrayal of the actual scenario. DIORAMA [Appendix A] uses PDAs and RFiD readers to collect information about the victim's position and this information is transmitted to the *Remote* server using the wireless LAN.

Disaster response activities are conducted for a short duration of time. This duration depends on the seriousness of the calamity. So disaster response systems are active and operational only during this period. Unlike regular wired and wireless networks, Disaster response systems like DIORAMA are setup and dismantled once the activity is over. Hence the requirement of the IT infrastructure in systems like DIORAMA is not

`

perennial. However, that does not undermine the need of high availability of these components during the actual times of operation. Rather they are very critical during the operational periods.

A complete recovery model for a disaster response system like DIORAMA is beyond the scope of this work. Hence here we provide the suggestive measures the administrators can take for different types of fault alarms triggered by DiNAR. For DIORAMA, whose availability requirements range from very high during the times of operation to almost zero once the evacuations are done, the recovery measures for IT related failures should involve sufficient amount of redundancy of hardware like wireless routers, 3G cards, remote clients like PDAs/Laptops, RFiD readers.

Below we discuss different fault types detected by DiNAR in our present implementation for DIORAMA and the recovery measures:

- Application Failure: At the disaster site, application failures could mean inability to read information from the RFiD readers. Hence if the correlation algorithm tags the RFiD reader interfacing application as root cause failure, then the administrator looking at this picture should instruct the paramedics at the disaster site to, restart the application on the tracking device. If this fault is accompanied with a non root cause alarm for the RFiD reader, then it is advisable to look at the operational status of the RFiD reader as well.
- Application Unresponsive: If any application is being reported as unresponsive, then its status should be monitored for 2-3 update cycles. This is to allow the application to become responsive again, as it is not an uncommon event for an application to not respond at certain intervals. If the alarm does not clear, the

`

administrator should direct the paramedics about the situation and restart this application.

- RFiD-Reader Down: Hard failures of the RFiD reader will lead to a root cause fault being shown in the DiNAR GUI. The administrator should immediately direct the paramedic to replace the concerned reader and restart the interfacing application to do the initial handshake. This is needed considering the importance of the RFiD reader in the whole setup of DIORAMA and also the fact that troubleshooting it will take more time in an already time critical application.
- Device Down: This alarm applies to a non responsive agent which is declared to be down by the DiNAR manager. Handling this alarm will be relatively different than the others as described above. It is due to the fact that this alarm is deduced due to an agent which did not respond for two update cycles followed by 1 mins wait time. This could have three possible causes:
    - Device failure (shutdown/reboot)
    - Agent Failure (Agent application crash)
    - Network failure. Unable to connect to the Internet.

  We will deal with the recovery steps for "Network Failure" later. From DIORAMA perspective, the recovery steps should involve:

    - Check to see if the device is up and running. Make sure the agent is running. This can be done by tailing the agent logs. The other method would be to check the DIORAMA server to see if the device is

`

reporting data. If so it is an agent failure and the agent needs to be restart.

- If the device is down physically, then it should be replaced with a backup device before beginning the troubleshooting.

Checking for the agent failure before the more severe possibility makes sure that the DiNAR operations does to hamper the core operations of DIORAMA.

- Cloud Unresponsive: A *CloudUnresponsive* alarm hints at a problem in the network connectivity. This is triggered when all the agents behind a particular cloud are down. The administrator should direct the paramedics to first replace the existing wireless gateway with a backup (having same SSID) and make sure all the devices are reconnected to the new wireless gateway. If this still does not clear the alarm, then each and every agent should be checked as explained above.

6.2. Remote Site

Remote site in DIORAMA is normally a command center with good wired network connectivity and high performance servers. Hence recovery measures in this site are not as complicated as in the disaster site. Failures in remote site would include:

- Application Down/Unresponsive
- Server failure
- Network failure

`

The failures on the remote server, once detected can be corrected by regular hardware/software/network recovery processes. It does not need any special instructions unlike components in the disaster site.

`

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Design of DiNAR was aimed to be an end to end solution for managing disaster response systems and provide real time fault detection. This thesis work was an effort to prove this concept. With its current architecture and implementation, DiNAR can also apply to other distributed application scenarios which have their elements distributed and use only internet to connect among themselves, with no overlays. One other example of such application is an Intel Health Guide [1] which is an in house patient monitoring system. Nevertheless, disaster management systems are still the most relevant applications for DiNAR.

The main contribution of this thesis work in designing DiNAR was the design and implementation of the DiNAR manager-agent architecture and agent initiated collection method. We implemented the analysis engine using the dependency graph algorithm which helped DiNAR from just being a collection schema to a solution.

Future Work:

- One of the wrong design choices of DiNAR was the use of external IP for grouping agents in a cloud. This mechanism is not flexible to accommodate the usage of cellular network for data connectivity. Hence new methods should be explored.
- Developing the agent for more platforms, mainly mobile device.

`

APPENDIX

DIORAMA OVERVIEW

DIORAMA is a system designed with an aim to expedite the EMS triaging process. It is a solution designed by the *Multimedia Networking Lab, University of Massachusetts, Amherst.* It uses the active RfID technology to track the victims after disaster and collects all the data about the victims, their locations and beams it to a server which is on the other side of the Internet.

Diorama System Architecture: Figure 18 shows the system architecture of DIORAMA. On a broad level DIORAMA can be visualized to be divided into two zones:

- Disaster Site: Where the actual disaster has occurred and where the Emergency MS triaging process will be carried out.
- Remote Site: The place where a server resides and collects all information from the disaster site.

The disaster site can be visualized as an open area which has been affected and has victims scattered around. This is shown in Figure 17. Each of the circular zones is one disaster site. Normal Emergency Medical Services (EMS) procedures [2] involves a triaging round where the paramedics arrive at the disaster site and triage the victims using paper tags. These tags contain information about the victim and his present status. Although this triaging technique has been in place since a long time, applying modern day Information Technology (IT) to it can revolutionize the whole process. This is what DIORAMA aims to achieve.

`

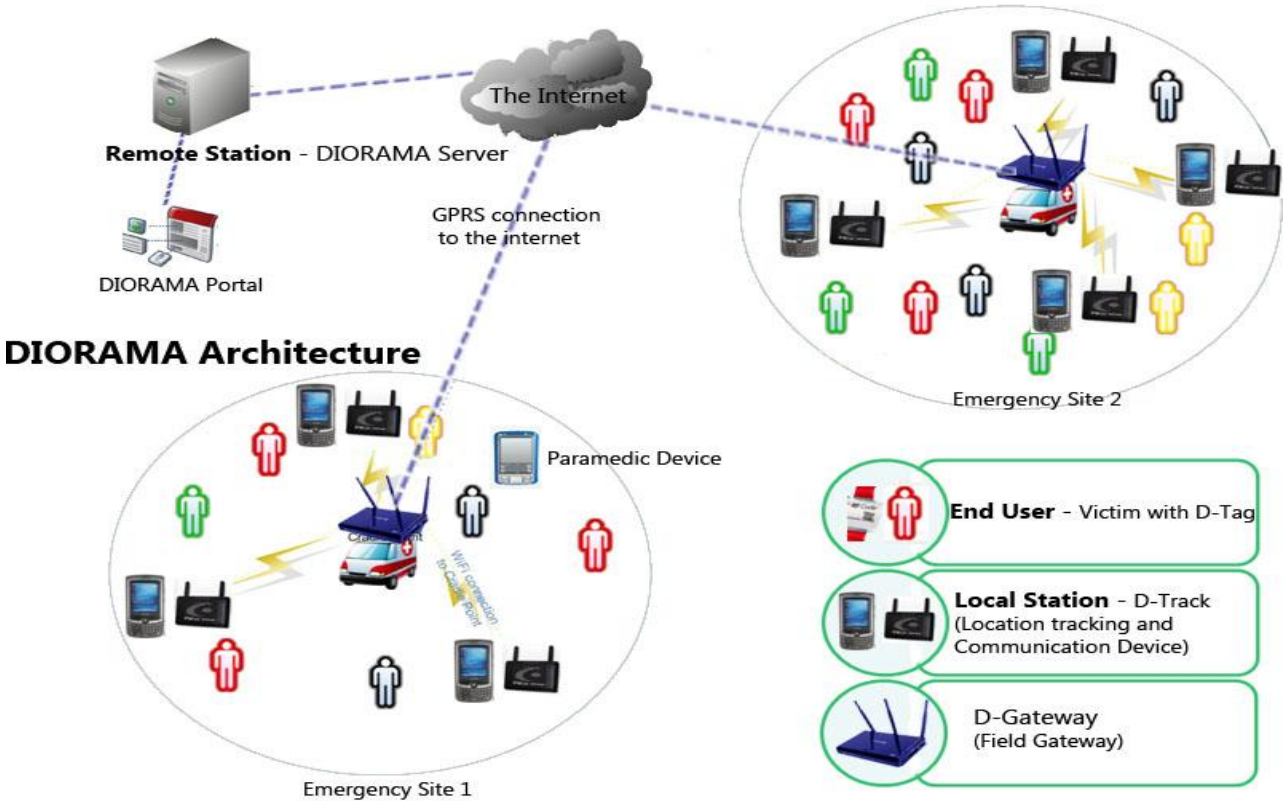

Figure 17 DIORAMA Overview

`

BIBLIOGRAPHY

[1] Intel Corporation, Intel Health Guide PHS6000, Product Guide.

[2] Southeastern Massachusetts EMS council. Manual: Policy and Procedures. pp. 36-37

[3] J. Case, M. Fedor, M. Schoffstall, J. Davin. RFC1157 - Simple Network Management Protocol (SNMP). (May 1990)

[4] R. Presuhn, J. Case, K. McCloghrie, M. Rose, S. Waldbusser. RFC3418 - Management Information Base (MIB) for SNMP (December 2002)

[5] Jürgen Schönwälder, Aiko Pras, Jean-Philippe Martin-Flatin. On the Future of Internet Management Technologies. In *IEEE Communications Magazine* (October 2003). pp. 90-97.

[6] J. Patrick Thompson. Web-Based Enterprise Management Architecture. In *IEEE Communications Magazine* (March 1998). pp. 80-86.

[7] Ramesh R. Rao, Jon Eisenberg, Ted Schmitt. THE ROLE OF IT IN MITIGATION, PREPAREDNESS, RESPONSE, AND RECOVERY. Book ISBN: 0-309-66744-5, National Academics (2007)

[8] Boris Gruschke. Integrated event management: Event Correlation using dependency graphs. In *Distributed Systems, Operations and Management* (1998).

[9] C. Robert Nelms. The Problem with Root Cause Analysis. In *Joint 8th IEEE HFPP / 13th HPRCT*. (2007). pp. 253-258

[10] DPS Telecom. Tutorial: Simple Network Management Protocol.

`

[11] Fummi. F, Quaglia. D, Stefanni. F, “Network Fault Model for Dependability Assessment of Networked Embedded Systems”. Computers, IEEE Transactions, Volume**:** 58, Issue: 5 On page**(s):** 620 – 633

[12] Deshpande. K, Ganz. A, “DiNAR: Health monitoring of IT systems in emergency response”, Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE, on page**(s):** 1699 - 1702

[14] W. Chen, N. Jain, S. Suresh. “ANMP: Ad Hoc Network Management Protocol”. IEEE Journal on Selected Area in Communications, VOL. 17, NO. 8, August 1999

[15] C.J. Chiang, et al. “Generic Protocol for Network Management Data Collections and Dissemination”, IEEE Military Communications Conference, October 2003.